Resource Allocation in Cloud Datacenters

476/576 Class Project: Team 06
Rand Perrin Black, Sean Bullis, Kira Kopcho, Morel Kopcho
Oregon State University, Corvallis, OR, USA, blackra,bulliss,kopchok,kopchom@oregonstate.edu

***Abstract*—With the growing demand for access to cloud resources, the problem of allocating resources to clients in real-time grows exponentially difficult. To manage the growth of demand on cloud service providers more efficient methods must be developed to ensure the quality of service expectations for clients are met. Specifically, three groups of algorithms have emerged as solutions to this problem space; namely the heuristic, meta-heuristic, and hybrid algorithms. There are many algorithms in each of these categories with different strengths and weaknesses in solving particular aspects of the allocation problem. This paper discusses the challenges faced in allocating resources to clients in large cloud data centers, explores each of the algorithm groups, and highlights an algorithm from each group used to tackle these problems. In addition, the implementation of two algorithms and the results of simulations of each of the implementations are discussed.**

I Introduction: Motivation and Objectives

Cloud computing refers to a paradigm where a cloud service provider (CSP) offers a pool of configurable computing resources. End-users interact with these resources by invoking and releasing them on a pay-per-use basis. One of the key advantages of cloud computing is that users are not required to have in-depth knowledge of the underlying virtual or physical machines, nor do they need to manage tasks intricately. Additionally, CSPs typically guarantee performance to end-users through service level agreements (SLAs), ensuring reliability and quality of service. This model allows for scalability, flexibility, and cost-effectiveness in deploying and managing applications and services.

The motivation for improving resource allocation in cloud computing arises from several key factors. Firstly, as cloud computing becomes increasingly prevalent, the associated power and thermal costs escalate exponentially. This necessitates efficient resource allocation strategies to minimize operational expenses while maintaining optimal performance. Secondly, the unpredictable fluctuations in demand pose challenges for optimizing resource utilization. CSPs must contend with varying workloads and strive to allocate resources effectively to meet dynamic demands. Moreover, CSPs face the delicate task of balancing the imperative to enhance cost efficiency and reduce energy consumption with the obligation to fulfill end-users' expectations and adhere to SLAs. Therefore, developing robust resource allocation mechanisms is crucial for ensuring reliable and sustainable cloud computing infrastructures.

II Background and Fundamental Concepts

A. Cloud Computing Architecture

Cloud computing takes on different forms. The three-tier architecture of cloud computing is one of the most common architectures. It encompasses distinct layers, each serving a specific function in the delivery of services and management of data. At the presentation tier, often referred to as the front end, users interact directly with the system, accessing the interface and visual elements that facilitate interaction with the application. This tier encapsulates what users see and interact with, shaping their experience and providing a means to input commands or retrieve information. Moving to the application tier, also known as the logic tier, this layer serves as the engine behind the scenes where the processing of information occurs. Here, algorithms, business logic, and data processing operations take place, orchestrating the functionality and behavior of the application based on user inputs and system requirements. Finally, at the data tier, or the back end, lies the infrastructure responsible for storing and managing data. This tier houses databases, file systems, and other storage mechanisms where data is persistently stored, organized, and retrieved as needed by the application. It serves as the foundation upon which the application operates.

In addition to the traditional three-tier architecture, there are several other common architectures and topologies present in the realm of cloud computing, each tailored to address specific needs:

- Microservices Architecture: Microservices architecture is a design approach where applications are broken down into smaller, loosely coupled services, each responsible for specific functionalities. These services are independently deployable and scalable. Microservices architecture promotes

modularization, enabling teams to develop, deploy, and update components independently.

- Hybrid/Multi Cloud Topology: Hybrid or multi-cloud topology involves the use of multiple cloud service providers or cloud environments, such as public, private, or hybrid clouds, to host an organization's workload. This approach offers redundancy, and scalability at the cost of more points of attack from a security standpoint.
- Edge Computing Topology: Edge computing topology decentralizes computation and data storage by placing resources closer to the point of data generation or consumption, typically at the edge of the network. This enables low-latency processing, real-time analytics, and bandwidth optimization for applications that require rapid response times or operate in bandwidth-constrained environments. Edge computing is particularly relevant for applications where minimizing latency and enhancing user experience are paramount.

B. Service Models

Beyond the topology of the system is the services being provided by those systems. CSPs offer a spectrum of services categorized primarily into Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). These service categories represent different levels of abstraction and management responsibilities, catering to varying needs and preferences of users:

- Infrastructure as a Service: IaaS provides users with virtualized computing resources over the internet. Users have access to scalable and on-demand infrastructure components, including virtual machines, storage, and networking resources. With IaaS, users retain control over the operating systems, applications, and data, allowing for greater flexibility and customization.
- Platform as a Service: PaaS offers a higher level of abstraction by providing a platform for developing, deploying, and managing applications without the complexity of underlying infrastructure management. PaaS offerings typically include development tools, middleware, databases, and runtime environments, enabling developers to focus on application development rather than infrastructure management.
- Software as a Service: SaaS delivers software applications over the internet, eliminating the need for users to install, maintain, or manage the software locally. Users access SaaS applications through web browsers, and the software is typically hosted and maintained by the CSP. SaaS offerings cover a wide range of applications, including productivity tools, customer relationship management systems, and collaboration platforms.

Before moving into specific instances of these services, a framework for determining quality of a cloud computing system is needed. The evaluation of cloud computing systems involves the assessment of various metrics to gauge performance, efficiency, and overall effectiveness. These evaluation metrics provide valuable insights into the functionality and capabilities of cloud-based infrastructures, enabling stakeholders to make informed decisions regarding deployment, optimization, and resource allocation. Evaluation metrics can be broadly categorized into objective-based metrics and performance metrics, each serving distinct purposes in evaluating different aspects of cloud computing systems. Objective-based metrics focus on assessing specific objectives or criteria to ensure that cloud services meet predefined goals and requirements. Performance metrics, on the other hand, delve into quantifying the performance characteristics of cloud systems to measure the system's operational efficiency and responsiveness to user demands. Objective-based metrics often include the following:

- Cost is a metric that applies to both parties involved. While service providers aim to maximize profit, end-users strive to minimize expenses. Effective cost management strategies are imperative for ensuring the financial viability and competitiveness of cloud services.
- Makespan refers to the total time it takes for a given set of tasks to complete. This is most heavily influenced by the scheduling algorithm used and is best minimized by the decided upon algorithm in the context of the tasks required to run.
- Scalability is the metric that measures the ability of the system to increase in size along with the number of end-users or tasks. This is a limiting factor for any kind of service provider.
- Load balance is the distribution of tasks as evenly as possible across available resources, ensuring optimal resource utilization and preventing bottlenecks or overloads in the system.
- Reliability assesses the likelihood that a given task can be completed without failure. This metric is of higher priority in the context of quality of service.

By considering these objective-based metrics, an end user can adequately determine if a service is correct for their needs, and stakeholders are able to determine if their service is of a competitive quality to others in the

market.

III Research Challenges

This section addresses the following challenges in context of research allocation in cloud computing. Note that these issues are not independent from each other and are deeply coupled increasing the difficulty of addressing one set of challenges without complicating or perturbing the issues faced in the other challenges.

- **Real-time Scheduling:** Real-time scheduling is a problem because as the size of the problem increases the complexity of the problem increases exponentially [1]. Thus, one must predict when a solution must be determined and predict the conditions of the system when that solution is needed to provide a fast and accurate result since a real-time solution at scale is unfeasible. Many users can request resources and terminate access at any time which makes this a highly dynamic system therefore good prediction models must be developed.
- **Quality of Service Balancing:** Cloud computing gives any client access to cloud resources if they are willing to pay. It is not guaranteed that each client is going to have the same metrics for what they consider quality service, therefore each client will need to have their quality of service metrics optimized as not to breach their service level agreement. These optimizations may be aimed at makespan, cost, energy use, reliability, or load-balance. This means that a one-sized-fits-all algorithm for resource allocation will not match each individual client demands.
- **Scalability:** As more users globally demand access to cloud network resources the complexity of the problem increases exponentially. The adoption of heuristic, meta-heuristic, and hybrid algorithms marks a shift from finding the best solution to finding a quasi-optimal solution in a reasonable amount of time. As the scale of resource allocation problems increase it is not clear that these solutions will remain viable, or whether the current topology and heirarchy of cloud data centers will be optimal. Considerations must also be made toward idle cloud resources and energy use as larger client loads will require more idle servers to buffer against potential surges in workload resulting in wasted power and increased carbon emissions.
- **Load Balancing** Resource utilization must be managed to ensure clients are not experiencing performance bottlenecks that hurt their quality of service, and that resources aren't sitting idle wasting energy and increasing the cost per unit work. With distributed systems all over the world and automated initialization and allocation of virtual machines the tracking of these resources becomes massively important. Ideally, system resources will be near maximum utilization without over-utilization but keeping them there with a dynamic workload is difficult especially when implementing solutions that are not inherently optimal including heuristic and meta-heurstic algorithms which aim to produce quasi-optimal solutions.
- **Energy Efficiency** When responding to client requests in real-time having resources already available is incredibly important, but each idle system waiting to be used wastes power and creates carbon emissions for zero accomplished work [2]. Challenges arise in optimizing the amount of available resources while minimizing non-utilized resources. Because of this, predictive models must be used to predict the potential incoming workload to ensure that just enough resources are available to handle the load, while a recovery system robust enough to quickly handle miss predictions to reduce QoS impact on clients.

IV Advances on the State-of-the Art: Solution Approaches

Resource allocation and task scheduling in the cloud is considered an NP-Hard problem, meaning it is both hard to solve the problem and hard to verify the solution of the problem. NP-hard problems increase complexity exponentially as the problem size increases.

Numerous optimization algorithms have been applied to the problem of resource allocation in the cloud. We explore three different categories of these optimization algorithms- Heuristic algorithms, Meta-heuristic Algorithms, and Hybrid Algorithms. Each algorithm has its strengths and weaknesses depending on what's being optimized and the problem size. This section will give a brief explanation of what each algorithm category is and an exploration of different algorithms within that category and their applications to resource allocation.

A. Heuristic Algorithms

Rather than an exhaustive search of the problem space for an optimal solution, heuristic algorithms aim to provide an approximate best solution within cost and timeframe constraints. They are derived from past performance information about the system that they are implemented on, and attempt to capture the relationship between the metrics being optimized, hardware utilization, and user workload patterns [3].

The advantages of heuristic algorithms include that they are able to run faster than traditional search algorithms, relatively simple to implement, and tend to be well-suited to online task scheduling problems [3]. Unfortunately, heuristic algorithms tend to be expensive in terms of proccessing time and storage cost. Furthermore, their performance weakens when there aren't many previous data to use, or the previous data do not follow a particular distribution [4].

In [5], the authors proposed an algorithm to minimize energy consumption and makespan in multi-cloud, heterogeneous networks. The algorithm, Energy-aware Task Allocation in Multi-Cloud Networks (ETAMCN), operates by first calculating the Expected Time to Completion (ETC) for each VM to find a set VMs that can accomodate the incoming task, then calculates the Energy Consumption (EC) of these VMs to find the one that will minimize energy consumption. ETAMCN was found to perform comparatively against Cloud Z-Score Normalization (CZSN) and Multi-Objective Scheduling with Fuzzy Resource utilization (FR-MOS) in terms of makespan; in terms of energy consumption ETAMCN outperformed CZSN by more than 6% and FR-MOS by approximately 3%.

In [6], the authors utilize game theory and inequality theory to propose the Static Mixed Nash Equilibrium (SM-NE) algorithm. SM-NE operates by allowing devices in fog computing networks to allocate tasks among nearby devices and the edge cloud to improve average completion time. When compared to the Myopic Best Response (MBR) algorithm, SM-NE was found to perform similarly in terms of performance gain. However, SM-NE requires less signalling overhead, as it only needs average system parameters, whereas MBR requires global knowledge of the system state.

B. Meta-Heuristic Algorithms

Meta-heuristic algorithms offer a versatile approach to solving complex optimization problems that lack clear, deterministic solutions. Unlike traditional heuristic methods, meta-heuristic algorithms do not rely on specific problem-solving strategies tailored to a particular domain. Instead, they operate at a higher level, employing general-purpose techniques to explore and navigate vast solution spaces in search of optimal or near-optimal solutions. These algorithms often involve exhaustive search processes, where they iteratively evaluate numerous potential solutions from large data pools to identify the most promising candidates. Commonly, meta-heuristic algorithms draw inspiration from natural phenomena or processes, such as evolutionary algorithms, simulated annealing, genetic algorithms, ant colony optimization, and particle swarm optimization, among others.

Advantages:

- Exact Solutions: Meta-heuristic algorithms have the capability to provide exact or near-exact solutions to complex optimization problems, making them suitable for scenarios where precision is paramount.
- Handling Large Solution Spaces: These algorithms excel at processing large or complex solution spaces, allowing them to effectively explore a wide range of possibilities and identify optimal solutions.

Drawbacks:

- Time-Consuming: Meta-heuristic algorithms often require a significant amount of time to converge towards optimal solutions, especially when dealing with complex or high-dimensional problem spaces. The iterative nature of these algorithms may lead to prolonged computation times.
- Resource Intensive: Due to their exhaustive search processes and iterative nature, meta-heuristic algorithms can be computationally intensive, demanding substantial processing resources to execute efficiently.

An example of a meta-heuristic algorithm is the Ant Colony Algorithm, which draws inspiration from the foraging behavior of ant colonies. The Ant Colony Algorithm emulates the foraging pattern observed in natural ant colonies, where ants explore their environment to locate food sources and communicate with each other through pheromone trails. ants traversing through a path, deposit a "pheromone" along the way. The amount of pheromone deposited on a path corresponds to the quality or desirability of that path. Ants tend to follow paths with higher concentrations of pheromones, as these paths are perceived as more favorable or promising. Over time, paths with higher pheromone levels attract more ants, reinforcing their attractiveness and increasing the likelihood of subsequent ants choosing those paths. Through repeated iterations, the Ant Colony Algorithm optimizes the search process, gradually converging towards a solution by effectively exploiting the collective intelligence and decentralized decision-making of an ant colony.

C. Hybrid Algorithms

Hybrid algorithms, by their namesake, are a combination of two or more algorithms to optimize a given objective (or objectives). Hybrid algorithms can be

a combination of meta-heuristic and heuristic algorithms or a combination of two or more meta-heuristic algorithms. In the literature surveyed about hybrid algorithms applied to resource and task scheduling, the hybrid algorithms used most commonly combined two or more meta-heuristic algorithms. The goal of hybridizing algorithms is to address shortcomings in individual algorithms such as low convergence or slow search speed. [7]

Of the hybrid algorithms studied- the most common hybrids were hybrids of the Genetic Algorithm, hybrids of the Particle Swarm Optimization (PSO) algorithm, and hybrids of the Ant Colony Optimization algorithm. An in-depth description of the Genetic Algorithm and the Particle Swarm Optimization algorithm can be found in VI-A.

In [8] a hybrid of PSO and the Artificial Bee Colony (ABC) algorithm was proposed to optimize the energy consumption of cloud data centers while not violating the service-license agreements of the end users. The ABC algorithm is based on the foraging behaviors of bees while PSO is based on swarm behaviors. ABC-PSO addresses the shortcomings of PSO with getting stuck in local optima (weaker exploration) and ABC's issue with having a weaker global search ability (weaker exploitation). The algorithm outperformed basic PSO and ABC in terms of energy consumption and throughput but did worse in terms of total execution time.

In [9] a hybrid of an improved max-min scheduler was combined with the ACO algorithm to minimize total makespan. Max-min schedulers prioritize large tasks over smaller tasks for scheduling. The improved variant of min-max is optimized based on the execution time of tasks rather than the completion time. In the ACO portion of the algorithm, the length of the paths is based on the execution time. The resulting task with the max execution time is scheduled to a resource (VM or Host) with an overall minimum completion time. Both the processing time and completion cost over the original improved Max-Min algorithm.

In [10] a hybrid of the Genetic Algorithm and an Energy-Conscious Scheduling model. The goal is to optimize makespan and energy consumption. The Genetic Algorithm prepares the sequences of tasks based on their priority, while ECS is used to assign tasks to processors. The fitness function of the genetic algorithm is based on energy consumption. The hybrid GAECS algorithm outperformed standard PSO and the ABC algorithm in terms of makespan and energy consumption.

Overall hybrid algorithms boast the ability to be efficiently adapted to multiple objectives and address issues that are present in non-hybrid meta-heuristic and heuristic algorithms. The downside of these hybrids however is they usually take a longer time to execute as and their computational complexity increases due to increased parameters to tune.

TABLE I
Comparison of Task Scheduling Algorithms

Algorithm	Category	Metric(s) Optimized	Algorithms Compared	Results
ETAMCN	Heuristic	Energy consumption, makespan	SZCN, FR-MOS	Outperformed in terms of energy consumption
SN-ME	Heuristic	Average completion time	MBR	Similar performance, less signalling overhead
PSO-ABC	Hybrid	Energy consumption	PSO, ABC	Outperformed in terms of energy consumption and throughput
ACO-Max-Min	Hybrid	Makespan	Max-Min	Outperformed in terms of processing time and completion cost
GAECS	Hybrid	Energy consumption, makespan	PSO, ABC	Outperformed in terms of makespan and energy consumption

V Unsolved Technical Challenges

Despite current research efforts in the area of resource allocation for cloud computing there are unsolved issues to be addressed. This section will discuss those issues as mentioned in literature.

- **System Profiling:** Currently, the way system performance is determined is through measuring metrics to extrapolate where a system bottleneck might be. This is important because cloud computing is ultimately a service in which an expectation of service quality is to be maintained. As distributed systems grow larger determining system performance through metrics becomes more vague and doesn't address the underlying problem of the system or how to improve it [11]. More fine-grained details of application and infrastructure performance must be obtained, but there exists a lack of relevant simulated streaming workloads required to charac-

terize how the fine grained details can be adjusted in order to tune system performance.

- **Task Consolidation:** Cloud computing networks can service a variety of different tasks from clients. These tasks will have different resource access patterns. Some applications resource access patterns may be more well suited to be paired with other types of access patterns such that resource contention for similarly required resources is minimized. Currently, there is no comprehensive research on what types of tasks can be paired with other types of tasks [4]. Tasks might include video streaming, graphics processing, data processing, etc. By optimally pairing tasks together better system performance can be achieved without additional infrastructure.
- **Resource-aware Scheduling:** It is a common misunderstanding that a system with some level of hardware resources will have those hardware resources available to provision to clients at all times. In reality, there may be background processes, system updates, or other overhead on the servers that reduce the available resources to clients. Without taking in the dynamic resource availability to clients, a system is prone to over-utilization and a degradation of performance will occur [11]. Resource-aware scheduling will allow for optimal task scheduling in the face of changing availability of resources. Specifically, when long-term tasks are running and overhead processes begin to run a remapping of tasks throughout the distributed network can occur to keep each task running smoothly.
- **Carbon Emission Management:** Various forms of green energy in the forms of wind, solar, water, and biomass are becoming more prevalent. Shifting task scheduling toward prioritizing resources running on servers powered by green energy will reduce the carbon footprint of the cloud data centers [11]. Predictive methods to determine available green energy clusters must be developed. Additionally, energy saving techniques are generally in conflict with QoS goals such as make-span and cost-efficiency as tasks may be allocated to clusters farther away or with fewer active nodes. Models must be developed to compare scheduling algorithms to determine the optimization of meeting service level agreements while minimizing usage of traditional fossil fuel based energy [11].

VI Case Study: Implementation and Evaluation

A. The Studied Techniques

For the implementation portion of the project, we chose to implement two meta-heuristic algorithms. As discussed earlier in this paper, meta-heuristic algorithms aim to iterate over a particular search area until an optimal solution is found. In the following section we will explore in depth the Particle Swarm Optimization (PSO) algorithm and the Genetic Algorithm and their applications to task scheduling.

1) Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population-based swarm-intelligence algorithm originally proposed by Kennedy and Eberhart in [12] in 1995. Originally tested on training neural network weights as an alternative to gradient descent, PSO has been applied as a method of solving numerous optimization problems-including the problem of task scheduling. As its name implies, PSO operates by using a swarm of particles to explore a given search space and obtain an optimum solution. Each particle's position in the search space represents a potential solution to the problem. These positions are randomly determined at initialization. At each generation of the algorithm, the position and velocity of the particle are updated based on the local and global optimum. Local optimum refers to the saved best position or "personal best" solution of an individual particle while global optimum refers to the best solution in the entire swarm [13]. Every iteration of PSO drives all particles in the swarm to converge around the global best solution, which is considered the optimal solution to the problem.

Implementation-wise, the basic PSO algorithm can be broken down into 4 main steps: 1) Evaluate the fitness of each particle, 2) Update the particle's velocity based on the global and local optimum, 3) Update the particle's position based on the applied velocity, 4) Update the global and local optimum based on each particle's new position. This process is repeated until a certain number of iterations are met or the global optimum does not change for multiple iterations [14]. Fig 1 presents a basic flowchart that shows the flow of the PSO algorithm.

The fitness and evaluation depend on which objective is trying to be optimized. In the area of task scheduling this can be makespan, execution time, cost, bandwidth, or energy efficiency. Each particle's fitness is evaluated using a fitness function, which determines how close the particle is to the best solution in the search space. The fitness function is the objective function of what needs to be optimized. For instance, if the goal was

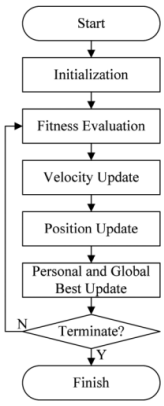

Fig. 1: Simple framework flowchart of the Particle Swarm algorithm from [14].

to optimize makespan then the fitness function would optimize the execution time of tasks as in [15]. The problem of task scheduling has been modeled as a single-objective and a multi-objective problem. Numerous different variants of the PSO algorithm have been applied to this problem, each meant to improve the base PSO algorithm to fit various problems [16].

Some of the variants of PSO that have been seen in the literature for task-scheduling are:

- **Jumping PSO** A variant of PSO that deals with a discrete solution space rather than continuous. The traditional model of applying velocity particles only works for continuous solutions. Instead, JPSO makes particles randomly "jump" to new locations based on current and future values of the particle's position.
- **Modified PSO** Variants of PSO that address certain shortcomings with the base algorithm such as getting trapped in local optima or long convergence times. These algorithms modify the structure of basic PSO to overcome its shortcomings.
- **Hybrid PSO** As mentioned earlier in the paper, PSO is often combined with other algorithms to improve its functionality and overcome drawbacks. Most often PSO gets combined with other meta-heuristic algorithms such as Ant Colony Optimization and the Genetic Algorithm.

PSO boasts many advantages to other meta-heuristic algorithms, the most prominent being its simplicity to implement. As with other swarm-intelligence-based algorithms- PSO has a fast search time and relatively low computational cost. However, while its simplicity gives it some advantages- it suffers from slow convergence times, especially with higher dimensional search spaces or more complex datasets. It also famously has an issue with getting trapped in local optima rather than converging on the global best solution. [17]. This is why basic PSO is often combined with other algorithms or modified to address these issues.

2) Genetic Algorithm

The Genetic Algorithm (GA) builds upon other evolutionary computation algorithms, spanning back as far as the 1960s. This includes Rechenberg's "evolution strategies" in 1965, as well as the work of Box and Friedman in the 1950s, Bledsoe, Bremermann, and Reed et al. in the 1960s, and numerous evolutionary biologists using computers to simulate evolution in controlled experiments [18].

GAs were invented by John Holland in the 1960s and developed by him and his students and colleagues at the University of Michigan throughout the 1970s. Contrary to prior evolutionary algorithms, Holland wanted to formally study the phenomenon of adaptation within nature, and implement such mechanisms into computer systems [18].

GAs aim to search the solution space by moving from a population of chromosomes (candidate solutions) to an optimal population via a process of "natural selection" that uses genetics-inspired operators of crossover, mutation, and inversion. Selection chooses chromosomes in the population to reproduce, with chromosomes with higher fitness values producing more offspring on average than less fit ones. Crossover exchanges portions of two chromosomes, roughly mimicking biological recombination between two single-chromosome organisms; mutation randomly changes the values of some locations in the chromosome; and inversion reverses the order of a contiguous section of the chromosome [18].

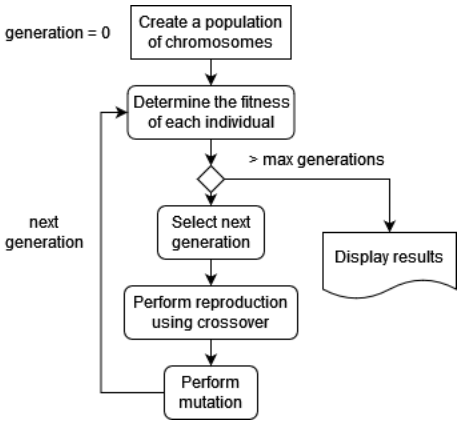

Fig. 2: Simple framework flowchart of the Genetic Algorithm, adapted from [19].

Figure 2 presents a high-level overview of GAs. The initial population is often created with random solutions/chromosomes. Subsequent generations are produced as a result of the cycle of determining fitness, selecting chromosomes, crossing over two chromosomes, and then performing mutation and/or inversion.

A fitness function is used to determine the quality of the individuals in the population with regard to an optimization objective. The fitness function will differ depending on the context of the optimization problem; in the context of resource allocation, it may be based on cost, energy usage, makespan, or execution time, among other metrics [20].

The selector is responsible for determining which individuals in the current generation will produce offspring. Many selection procedures perform fitness-proportionate selection, meaning that individuals with higher fitness values are selected with higher probability. However, there also exist non-parametric selection procedures that only examine the rank ordering of individuals' fitness values [21].

Crossover is the most noteworthy operation in GAs. Individual components or "genes" are split and recombined between two parent chromosomes. These parents may be swapped at a single point, between two points, or independently for each gene [21].

Mutation is used to prevent premature convergence of the GA. After several generations, it is possible for the selector to drive the values of particular genes to a single value. The role of mutation, then, is to prevent the loss of a given value of a gene or "allele". Depending on the type of mutation procedure chosen, genes in the chromosome will have their value flipped, or two genes will swap places with a set probability [22].

Beyond the standard genetic algorithm, variants of GA have been developed to be more exploitative in their search. Some of these variants include:

- **Elitism** During the selection phase, the fittest individuals are directly injected into the next generation, resulting in "better" solutions persisting across generations. [21].
- **Steady-State GA** Rather than creating a new generation all at once, this variant iteratively generates a few offspring, assesses their fitness, and then add them to the population by culling an equal number of existing individuals [21].
- **Tree-Style Genetic Programming** Genetic programming (GP) uses meta-heuristics to find highly fit computer programs; in this case using trees as its representation. During its crossover phase, the variant of GA used crosses the selected parents with 90% probability, otherwise it selects and directly copies one parent to the next generation [21].

These variants, and GAs as a whole, provide a number of advantages. Namely, they are conceptually simple, requiring no gradient information to operate; they are broadly applicable, as they can be applied to any problem that can be formulated as a function optimization problem; and they are robust to dynamic changes in the environment, as in most cases the population need not be reinitialized as the environment changes [23]. However, GAs often suffer from premature convergence, meaning that eventually the GA contrains the population to copies of the same individual [21].

B. Evaluation and Analysis

In our implementation of PSO and the Genetic Algorithm for task scheduling, we chose to use CloudSim. CloudSim is a free, open-source tool that allows users to simulate a data center environment. It provides a solution for studying processes in the cloud without having to gain access to actual hardware or requiring individuals to write their simulation engine. There are other simulators such as MATLAB and NS3/NS2-based simulators such as GreenCloud - however, CloudSim is one of the most complete and most well-documented of the simulators presented in the papers we surveyed. CloudSim is developed by the CLOUDS Lab at the University of Melbourne. For our project, we chose to use CloudSim version 3.0.3 since most of the tutorials are geared toward that specific version. Within the CloudSim library are numerous example codes showing users how to properly set up a data center with certain parameters. We utilized these example codes as a framework to build our implementations.

A quick explanation of some of the terminology associated with CloudSim:

- **Cloudlet** A model of a task in the cloud, parameterized by computational requirements such as length, processing units required to run the task, and resource utilization.
- **VM** A model of a Virtual Machine that is controlled by the data center host. It has parameters such as memory, processor cores, storage size, and a resource scheduling policy to handle cloudlets.
- **Host** A model of the actual hardware/physical resource that VMs and Cloudlets are executed on. Its parameters are available memory, processing cores, storage, and an allocation policy for sharing memory, processing power, and bandwidth among VMs. [24]

The most important user-expanded class in CloudSim

is the data center broker. The broker is responsible for mapping Cloudlets to available VMs- i.e. it is what does the task scheduling and resource assignment in the simulation. By default CloudSim uses First-Come-First-Serve (FCFS) for scheduling VMs to Cloudlets- however, this produces poor results when the number of Cloudlets is large. We extended the broker class to implement the PSO and GA respectively to schedule tasks to VMs.

We chose to keep the parameters of the Host, VMs, and Cloudlets common among our simulations so we could compare the algorithms as directly as possible. We used only 1 host to store our VMs and gave it 64 cores of processing power and a total of 2,356,230 MIPS. This is based on the specs for a Ryzen Threadripper 3900x processor. We also gave the host 640 GB of RAM and 1 TB of storage. On the host, we run 5 separate VMs. Table II shows the simulation parameters for each VM. Each VM had varied numbers of MIPS to test how well each algorithm scheduled tasks across VMs with non-uniform resources. All Cloudlets had 1 processing element and we randomly varied the length (in MIPS) from 100 to 1000 to simulate the variability of tasks submitted for processing in the cloud. We chose to use 100, 500, 1000, 1500, and 2000 Cloudlets to evaluate the ability of the algorithms to perform task scheduling that optimizes makespan. Each algorithm was run 3 times with each number of Cloudlets.

TABLE II
Virtual Machine Specifications

	V1	V2	V3	V4	V5
RAM (MB)	512	512	512	512	512
Image Size (MB)	1000	1000	1000	1000	1000
CPU Cores	2	2	2	2	2
Bandwidth (MB/s)	1000	1000	1000	1000	1000
MIPS	500	100	1000	1500	2500

1) PSO Setup

We implemented a basic version of PSO that minimized makespan based on the methods described in [15] and [25]. The fitness function used calculates the predicted makespan of each cloudlet based on a matrix of saved execution times for each combination of cloudlets on each VM. The maximum finish time gets returned as the fitness of each particle. For our final results, we used a swarm size (number of particles) of 100 and an iteration count of 1000. Originally we used 30 particles and 10000 iterations, but the convergence rate was very slow with those parameters and also they didn't match what was in the literature.

As discussed in [15], the cognitive and social coefficients to determine the velocity of a particle were set to 2. The inertia weight was updated on every iteration by taking the maximum inertia value minus the current inertia value w_i and dividing it by the number of iterations. The equations used for updating the velocity and position of the particle are shown below.

$$v_i = w_i * v_i + 2 * r_1(p_{best} - p_{curr}) \\ +2 * r_2(g_{best} - p_{curr}) \\ p_i = p_i + v_i$$

Where p_{best} is the personal best position of a particle, p_{curr} is the current position, g_{best} is the global optima and r_1 and r_2 are random vectors.

2) GA Setup

We also implemented GA to minimize makespan, with the fitness function used operating similarly to the function used in the simulation of PSO. To maintain consistency with the PSO simulation, the GA used a population size of 100, and iterated for 1000 generations.

The GA implementation uses a roulette-wheel selector, which is a fitness-proportional selector. The selection probability $P(i)$ is calculated using the fitness value f_i of individual i, with selecting n individuals from a given population equivalent to playing n times on the roulette wheel [26]. Equation 1 shows the equation used to calculate selection probability.

$$P(i) = \frac{f_i}{\sum_{j=0}^{N-1} f_j} \tag{1}$$

For the crossover procedure, we used uniform crossover, as it is more exploitative than other crossover procedures. Algorithm 1 provides an outline of the uniform crossover algorithm [21].

Algorithm 1 Uniform Crossover

```
p ← probability of swapping an index
v⃗ ← first vector ⟨v_1, v_2, ..., v_l⟩ to be crossed
w⃗ ← second vector ⟨w_1, w_2, ..., w_l⟩ to be crossed
for i from 1 to l do
    if p ≥ random number chosen uniformly from 0.0 to 1.0 inclusive then
        Swap the values of v_i and w_i
    end if
end for
return v⃗ and w⃗
```

We used swap mutation for the mutation procedure, which simply swaps two genes within the chromosome with a fixed probability.

3) Results and Analysis

We compared the implementations on average makespan, execution time, and throughput. In addition to PSO and GA, we also tested CloudSim's FCFS algorithm as a baseline to compare against. The following sections compare the results across each of the metrics with the values graphed being the median values across the three trials.

- **Average Makespan** Both PSO and GA vastly outperformed FCFS, with GA having slightly lower average makespans than PSO. At 100 cloudlets, PSO and GA have average makespans of 350.45 seconds and 307.75 seconds respectively, which are roughly a third of FCFS' average makespan of 1094.40 seconds. As the number of cloudlets is increased up to 2000, this ratio becomes closer to half of FCFS' average makespan of 20384.44 seconds, as compared to PSO and GA with average makespans of 12427.72 seconds and 10157.81 seconds respectively. Figure 3 graphs the average makespan for each algorithm with a varying number of cloudlets.

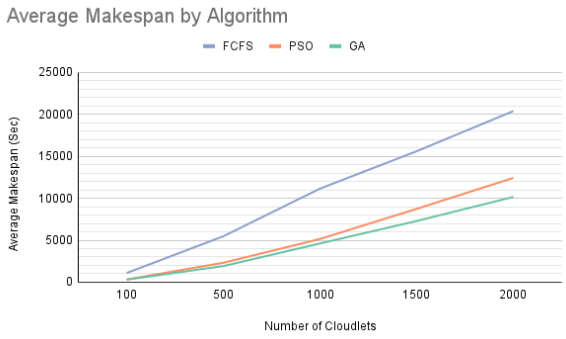

Fig. 3: Comparison of average makespan by algorithm

- **Execution Time** The execution times for all three algorithms increased exponentially with increasing number of cloudlets. That being said, PSO and GA again vastly outperformed FCFS. As with average makespan, at the lower end of cloudlets PSO and GA had execution times that were roughly one-third of FCFS' (35034.70 seconds and 30764 seconds as opposed to 109430.22 seconds); at the upper end of cloudlets the execution times were closer to one-half that of FCFS (24843158.97 seconds and 20315418.97 seconds as opposed to 40768682.19 seconds). Figure 4 graphs the execution time for each algorithm against a varying number of cloudlets.

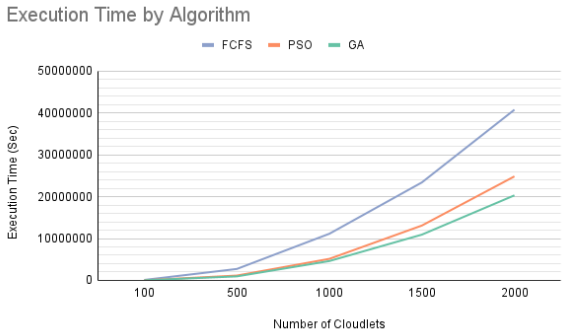

Fig. 4: Comparison of execution time by algorithm

- **Throughput** Once more, PSO and GA outperformed FCFS, maintaining much higher throughput across all numbers of cloudlets. Interestingly, the throughput for FCFS remained constant at 0.02. Throughput for PSO and GA starts comparatively high at 0.11 and 0.21 respectively, and declines towards the throughput for FCFS, with 0.03 and 0.04 respectively at 2000 cloudlets. Figure 5 graphs the throughput for each algorithm across varying numbers of cloudlets.

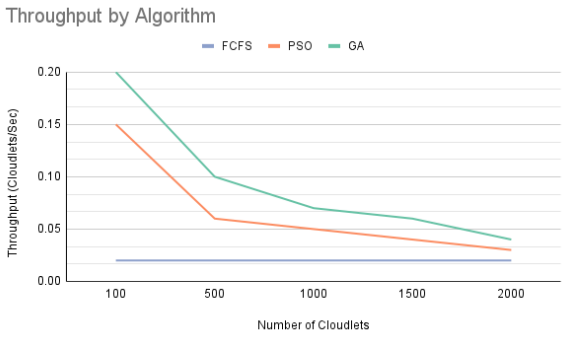

Fig. 5: Comparison of throughput by algorithm

VII Conclusion

We have discussed the challenges for resource allocation in cloud computing and motivated a discussion on why this is an important problem to solve. Finding real time solutions is too time expensive and infeasible as the size of the problem grows. Heuristic, meta-heuristic and hybrid algorithms are being explored to reach quasi-optimal solutions in a much faster time. Two such algorithms are the Genetic Algorithm (GA) and the Particle Swarm Optimization (PSO), both of which iterate on a set of randomized initial solutions to converge to an optimal solution space, and both of which are meta-heuristic.

PMO models a swarm of particles and uses position and velocity parameters to select for solutions that are better than the previous iterations, while de-selecting for solutions that are worse. PMO is generally an easy algorithm to implement and relatively low cost computationally, but suffers from long convergence times when the solution space is large. GA uses principles from genetics to evolve generations that converge to an optimal solution space. Genes are generated by mapping resources to tasks and random chromosomes are created by groups of randomized genes. Crosses and mutations are applied by iterating over the chromosome generations to select for the best solutions. GAs are highly applicable to most any optimization problem and robust to dynamic changes in the environment but suffer from premature convergence, such that the algorithm can get stuck in a non-optimal solution space when the chromosome population becomes too similar.

Based on the simulation results, GA has the best makespan time, with PSO being just slightly behind that. In comparison, these are much faster than the first-come first-serve (FCFS) approach which we compared against as baseline. The throughput for GA was higher at for smaller cloudlet counts but GA converged with PSO for larger cloudlet counts > 2000. The degree of imbalance for both GA and PSO was similar across the range of 100 to 2000 cloudlets, and both performed significantly better than (FCFS) at smaller cloudlet counts, and performed slightly better at larger cloudlet counts. As cloudlet count increases, the execution time for GA was slightly better than PSO, both of which were significantly better than FCFS.

Ultimately, both GA and PSO perform comparably across cloudlet count, and both perform significantly better than FCFS. Each have their own strengths and weaknesses that tend to be problem specific, and hybrid implementations of these algorithms can highlight the strengths of each while mitigating the weaknesses. It can be determined that heuristic, meta-heuristic, and hybrid algorithms improve the system performance of cloud computing in the context of resource allocation. Looking forward, future work would include comparisons between a hybrid implementation of GA and PSO in comparison against each of them individually.

VIII Group Member Contributions

- Kira Kopcho: Implemented PSO in CloudSim for Cloudlet scheduling, Survey/Research of hybrid algorithms, Introduction of CloudSim and discussion of CloudSim parameters in VI-B, in-depth discussion of PSO in VI-A, presentation of PSO in implementation presentation.
- Morel Kopcho: Implemented GA in CloudSim, presentation of implementation of GA in implementation presentation, survey/research of heuristic algorithms in survey presentation and in IV, in-depth discussion of GA in VI-A, graphs and results in VI-B.
- Rand Black: Wrote introduction and background sections. Researched/Surveyed meta-heuristic algorithms. Researched and presented on foundational cloud computing architectures and service models.
- Sean Bullis: Wrote the abstract, research challenges, unsolved technical challenges, and conclusions sections. Researched the genetic algorithm in depth and presented on it.

References

[1] *KSII Transactions on Internet and Information Systems*, vol. 14, no. 7, Jul. 2020. [Online]. Available: http://dx.doi.org/10.3837/tiis.2020.07.005

[2] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Energy-efficient resource allocation and provisioning framework for cloud data centers," *IEEE Transactions on Network and Service Management*, vol. 12, no. 3, pp. 377–391, Sep. 2015. [Online]. Available: http://dx.doi.org/10.1109/TNSM.2015.2436408

[3] N. Chauhan, N. Kaur, K. S. Saini, S. Verma, A. Alabdulatif, R. A. Khurma, M. Garcia-Arenas, and P. A. Castillo, "A systematic literature review on task allocation and performance management techniques in cloud data center," *arXiv preprint arXiv:2402.13135*, 2024.

[4] A. Hameed, A. Khoshkbarforoushha, R. Ranjan, P. P. Jayaraman, J. Kolodziej, P. Balaji, S. Zeadally, Q. M. Malluhi, N. Tziritas, A. Vishnu, S. U. Khan, and A. Zomaya, "A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems," *Computing*, vol. 98, no. 7, p. 751â774, Jun. 2014. [Online]. Available: http://dx.doi.org/10.1007/s00607-014-0407-8

[5] S. K. Mishra, S. Mishra, A. Alsayat, N. Z. Jhanjhi, M. Humayun, K. S. Sahoo, and A. K. Luhach, "Energy-Aware Task Allocation for Multi-Cloud Networks," *IEEE Access*, vol. 8, pp. 178 825–178 834, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9205994/

[6] S. Jošilo and G. Dán, "Decentralized algorithm for randomized task allocation in fog computing systems," *IEEE/ACM Transactions on Networking*, vol. 27, no. 1, pp. 85–97, 2018.

[7] N. Arora and R. Banyal Kumar, "Hybrid scheduling algorithms in cloud computing: a review," *International Journal of Electrical and Computer Engineering*, vol. 12, no. 1, pp. 880–895, 2022.

[8] J. Meshkati and Safi-Esfahani, "Energy-aware resource utilization based on particle swarm optimization and artificial bee colony algorithms in cloud computing," *The Journal of Supercomputing*, vol. 75, pp. 2455â–2496, 2019.

[9] N. S. Ghumman and R. Kaur, "Dynamic combination of improved max-min and ant colony algorithm for load balancing in cloud system," in *2015 6th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2015, pp. 1–5.

[10] P. Pirozmand, A. A. R. Hosseinbaldi, M. Farrokhzad, S. Mirkamali, M. Sadeghilalimi, and A. Slowik, "Multi-

objective hybrid genetic algorithm for task scheduling problem in cloud computing," *Neural Computing and Applications*, vol. 33, p. 13075â13088, 2021. [Online]. Available: https://link.springer.com/article/10.1007/s00521-021-06002-w

[11] X. Liu and R. Buyya, "Resource management and scheduling in distributed stream processing systems: A taxonomy, review, and future directions," *ACM Computing Surveys*, vol. 53, no. 3, p. 1â41, May 2020. [Online]. Available: http://dx.doi.org/10.1145/3355399

[12] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[13] X. Hu, "PSO Tutorial," accessed 2024-03-14. [Online]. Available: http://www.swarmintelligence.org/tutorials.php

[14] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches," *ACM Comput. Surv.*, vol. 47, no. 4, pp. 1–33, 2015. [Online]. Available: https://doi.org/10.1145/2788397

[15] H. S. Al-Olimat, M. Alam, R. Green, and J. K. Lee, "Cloudlet scheduling with particle swarm optimization," in *2015 Fifth International Conference on Communication Systems and Network Technologies*, 2015, pp. 991–995.

[16] M. Masdari, F. Salehi, M. Jalali, and M. Bidaki, "A survey of PSO-based scheduling algorithms in cloud computing," *Journal of Network and Systems Management*, vol. 25, no. 1, pp. 122–158, 2017. [Online]. Available: https://oregonstate.idm.oclc.org/login?url=https://www.proquest.com/scholarly-journals/survey-pso-based-scheduling-algorithms-cloud/docview/1860051769/se-2?accountid=13013

[17] A. G. Gad, "Particle swarm optimization algorithm and its applications: A systemic review," *Archives of Computational Methods in Engineering*, vol. 29, pp. 2531–2561, Apr. 2022. [Online]. Available: https://link.springer.com/article/10.1007/s11831-021-09694-4#citeas

[18] M. Mitchell, *An introduction to genetic algorithms*, ser. Complex adaptive systems. Cambridge, Mass: MIT Press, 1996.

[19] S. Kaur and A. Verma, "An Efficient Approach to Genetic Algorithm for Task Scheduling in Cloud Computing Environment," *International Journal of Information Technology and Computer Science*, vol. 4, no. 10, pp. 74–79, Sep. 2012. [Online]. Available: http://www.mecs-press.org/ijitcs/ijitcs-v4-n10/v4n10-9.html

[20] P. Kumar and A. Verma, "Independent task scheduling in cloud computing by improved genetic algorithm," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 2, no. 5, 2012.

[21] S. Luke, *Essentials of metaheuristics: a set of undergraduate lecture notes; Online Version 2.0*, 2nd ed. S.l.: Lulu, 2013.

[22] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, pp. 65–85, 1994.

[23] S. N. Sivanandam and S. N. Deepa, *Introduction to genetic algorithms*. Berlin ; New York: Springer, 2007.

[24] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011. [Online]. Available: http://www.buyya.com/papers/CloudSim2010.pdf

[25] Z. Yang, X. Qin, W. Li, and Y. Yang, "Optimized task scheduling and resource allocation in cloud computing using PSO based fitness function," *Information Technology Journal*, vol. 12, no. 23, pp. 7090–7095, 2013. [Online]. Available: https://scialert.net/abstract/?doi=itj.2013.7090.7095

[26] F. Wilhelmstötter, "Jenetics Library User's Manual 7.2," *URL: http://jenetics. io*, 2021.